Chapter 10
A Beautiful Theorem

Francisco Antonio Doria and Carlos A. Cosenza

Dedicated to Jair Minoro Abe for his 60th birthday

A thing of beauty is a joy for ever:
Its loveliness increases; it will never
Pass into nothingness

John Keats, Endymion

Abstract We first present Maymin's Theorem on the existence of efficient markets; it is a result that connects mathematical economics and computer science. We then introduce O'Donnell's algorithm for the solution of NP-complete problems and the concept of almost efficient markets; we state the main result, which is: given a metamathematical condition, there will be almost efficient markets. We then briefly discuss whether changing the underlying logical framework we would be able to change the preceding results.

Keywords Maymin's theorem · Efficient markets · O'Donnell's algorithm · Almost efficient markets

10.1 Prologue

Beauty in mathematics has many sources. One of them, the discovery of links between areas that seemed at first so far away. That's the case of Maymin's theorem on efficient markets: it is linked as in the gesture of a magician to complexity theory in computer

Partially supported by CNPq, Philosophy Section; the first author is a member of the Brazilian Academy of Philosophy.

F.A. Doria (✉) · C.A. Cosenza
Advanced Studies Research Group, HCTE, Fuzzy Sets Laboratory,
Mathematical Economics Group, Production Engineering Program,
COPPE, UFRJ, P.O. Box 68507, Rio Rj 21945–972, Brazil
e-mail: fadoria63@gmail.com

science. An apt paraphrasis for Maymin's result would be—there are weakly efficient markets if and only if a given major question in computer science is trivialized.

The major question is, of course, the *P* versus *NP* question.

The proof is simple, once we see that markets (in Maymin's sense) are naturally coded as Boolean formulae.

The present paper first sketches that construction. We then fill in the required details and prove Maymin's theorem. After that we construct O'Donnell's algorithm, show that it is near–polynomial given reasonable conditions, and then with the help of that algorithm we define "almost Maymin" efficient markets. We then prove the existence of almost Maymin efficient markets, again given a reasonable metamathematical hypothesis.

Main sources for these results are [3, 4]. And let's repeat here our motto: *A thing of beauty is a joy forever.* So is Maymin's theorem.

10.2 Theme

Think of this paper as a set of variations over a theme found elsewhere. We will pick up our main theme from two sources, Maymin's original paper [4] and a summary of it made in a recent paper by the author (with NCA da Costa) [2, 3, 5].

A Brief Scenario

We start here from this recent intriguing result by Maymin [4]. We use a modified, restricted version of Maymin's construction (but there is no loss of generality in our construction.) The concepts we require are that of a *Maymin market*, soon to be clarified.

Roughly, a Maymin market is a market coded by a Boolean expression, as we will see. Despite this very precise identification, the object we consider is quite general. Basically we are going to make some move in the market. Our move now is determined by a series of k previous moves. More precisely:

Definition 10.2.1 • A k-run policy σ_k, k a positive integer, is a series of plays (b for buy and s for sell) of length k. There are clearly 2^k possible k-run policies.

- A map v from all possible k-run policies into $\{0, 1\}$ is a valuation; we have a "gain" iff $v(\sigma_k) = 1$; a "loss" otherwise.
- A policy is *successful* if it provides some gain (adequately defined); in that case we put $v(\sigma_k) = 1$. Otherwise $v(\sigma_k) = 0$. □

There is a natural map between these objects and k-variable Boolean expressions (see below), if we take that $v(\sigma_k) = 1$ means that σ_k is satisfiable, and 0 otherwise. We say that a *market configuration* (k-steps market configuration, or simply k-market configuration) is coded by a Boolean expression in disjunctive normal form (dnf).

That map between k-market configurations and k-variable Boolean expressions in dnf can be made 1–1.

The financial game for our simplified market is simple: we wish to discover the fastest way to algorithmically obtain a successful k-market configuration, given a particular market (i.e., a given k-variable Boolean expression).

Finally the k-market configurations are Maymin–efficient (see below) if v can be implemented by a poly algorithm.

Clearly there is a general polynomial procedure to do it if and only if $P = NP$. From what we know about the P versus NP question,[1] in particular cases we can of course find polynomial procedures, but it is unknown whether there are general procedures that are polynomial.

Stretto

Maymin restricts his analysis to the so-called "weakly efficient" markets. Since he adds the condition that there is a time-polynomial algorithmic procedure to spread the data about the market, we name *Maymin–efficient markets* those markets, where (we stress) $v(\sigma_k)$ is computed by a time-polynomial Turing machine (or poly-machine).

So the existence of general poly procedures characterizes the market as *Maymin efficient*. We can therefore state Maymin's theorem:

Proposition 10.2.1 *Markets are (Maymin) efficient if and only if $P = NP$.* □

Now we put: markets are *almost Maymin efficient* if and only if there is an O'Donnell algorithm to determine its successful policies [3]. Then:

Proposition 10.2.2 *If $P < NP$ isn't proved by primitive recursive arithmetic then there are almost Maymin efficient markets.* □

We are now going to expand these brief remarks into a detailed proof of Maymin's theorem, and then add to it some spice of our own.

10.3 Theme and Variations

The main motive is very simple: we are going to code Maymin–efficient markets as Boolean expressions. This is the main trick. But how do we proceed?

We first require a classical result by Emil Post [6]. The 2^k binary sequences naturally code integers from 0 to $2^k - 1$; more precisely, from:

$$000\ldots00,\ k \text{ digits},$$

to:

$$111\ldots11,\ k \text{ digits}.$$

Fix one such coding; a k-digit binary sequence is seen as a sequence of truth values for a Boolean expression Ξ_k. After we test the Boolean expression with one

[1] Actually we know very little.

specific line of truth values, we see whether that particular line satisfies or doesn't satisfy Ξ_k.

Proposition 10.3.1 *Let ξ_k be a binary sequence of length 2^k. Then there is a Boolean expression Ξ_k on k Boolean variables so that ξ_k is its truth table.*

(We take 1 as "true" and 0 as "false."). The idea of the proof goes as follows. Notice that the Boolean expression:

$$\neg p_1 \wedge p_2 \wedge p_3 \wedge \neg p_4 \wedge \neg p_5$$

is satisfied by the binary 5-digit line:

$$01100$$

(When there is a $\neg$ in the conjunction put 0 in the line of truth-values; if not put 1.)

The line 01100 satisfies the Boolean conjunction above, and no other 5-digit line will satisfy it, that is, it has a truth table where a single line of truth values satisfies it—the truth table has a single 1 and is zero for all remaining truth value lines.

In order to obtain a truth table with just two 1's, we construct the conjuncts as above that have the desired lines and then get the expression which is the disjunction of those conjuncts. That is the idea in the proof of Post's theorem.

Trivially every k-variable Boolean expression gives rise to a 2^k-length truth table which we can code as a binary sequence of, again, size 2^k bits. The converse result is given by Post's theorem.

Proof of Post's theorem, a sketch: Consider the k-variable Boolean expression:

$$\zeta = \alpha_1 p_1 \wedge \alpha_2 p_2 \wedge \cdots \wedge \alpha_k p_k,$$

where the α_i are either nothing or $\neg$. Pick up the line of truth values $\zeta' = \alpha_1 \alpha_2 \ldots \alpha_k$, where "nothing" stands for 1 and $\neg$ for 0. ζ' satisfies ζ, while no other line of truth values does. Our Boolean expression ζ is satisfied by ζ' and by no other k-digit line of truth values.

The disjunction $\zeta \vee \xi$ where ξ is a k-variable Boolean expression as ζ, is satisfied by (correspondingly) two lines of truth values, and no more. And so on.

The rigorous proof of Post's theorem is by finite induction. □

Now:

Definition 10.3.1 The Boolean expression in dnf ζ is identified to a Maymin k-market configuration. □

Then:

Proposition 10.3.2 *There are Maymin–efficient markets if and only if $P = NP$.*

Proof Such is the condition for the existence of a poly algorithmic map v. □

10.4 The O'Donnell Algorithm

We are now going to describe O'Donnell's algorithm [2, 3, 5]; the O'Donnell algorithm is a quasi-polynomial algorithm for SAT.[2] We require the so-called BGS set of poly machines and f_c, which is the (now recursive) counterexample function to $[P = NP]$ (See [1, 3] for details.)

Remark 10.4.1 A BGS machine is a Turing machine $\mathsf{M}_n(x)$ coupled to a clock that stops the machine when it has operated for $|x|^p + p$ steps, where x is the binary input to the machine and $|x|$ is its length in bits; p is an integer ≥ 1. Of course the coupled system is a Turing machine. All machines in the BGS set are poly machines, and given any poly machine, there will be a corresponding machine in BGS with the same output as the original poly machine. □

Remark 10.4.2 f_c is the recursive counterexample function to $P = NP$. To get it:

- Enumerate all BGS machines in the natural order (one can do it, as the BGS set is recursive).
- For BGS machine P_n, $\mathsf{f}_c(n)$ equals the first instance of SAT which is input to P_n and fails to output a satisfying line for that instance of SAT. □

O'Donnell's algorithm is very simple: we list in the natural ordering all BGS machines. Given a particular instance $x \in$ SAT, we input it to $\mathsf{P}_1, \mathsf{P}_2, \ldots$ up to the moment when the output is a satisfying line of truth values. When we compute the time bound to that procedure, we see that it is near polynomial, that is, the whole operation is bounded by a very slow-growing exponential.

Now some requirements:

- We use the (fixed) enumeration of finite binary sequences

$$0, 1, 00, 01, 10, 11, 000, 001, 010, 011, \ldots.$$

  If FB denotes the set of all such finite binary sequences, form the standard coding $FB \mapsto \omega$ which is monotonic on the length of the binary sequences.
- We use a binary coding for the Turing machines which is also monotonic on the length of their tables, linearly arranged, that is, a 3-line table s_1, s_2, s_3, becomes the line $s_1 - s_2 - s_3$.
  We call such monotonic codings *standard codings*.
- We consider the set of all Boolean expressions in cnf,[3] including those that are unsatisfiable, or totally false. We give it the usual coding which is 1–1 and onto ω.
- Consider the poly Turing machine $\mathsf{V}(x, s)$, where $\mathsf{V}(x, s) = 1$ if and only if the binary line of truth values s satisfies the Boolean cnf expression x, and $\mathsf{V}(x, s) = 0$ if and only if s doesn't satisfy x.

[2] Actually we deal with a slightly larger class of Boolean expressions.

[3] Conjunctive normal form.

- Consider the enumeration of the BGS [1] machines, $\mathsf{P}_0, \mathsf{P}_1, \mathsf{P}_2, \ldots$.[4]

We start from x, a Boolean expression in cnf binarily coded:

- Consider x, the binary code for a Boolean expression in cnf form.
- Input x to $\mathsf{P}_0, \mathsf{P}_1, \mathsf{P}_2, \ldots$ up to the first P_j so that $\mathsf{P}_j(x) = s_j$ and s_j satisfies x (that is, for the verifying machine $\mathsf{V}(x, s_j) = 1$).
- Notice that there is a bound $\leq j = \mathsf{f}_c^{-1}(x)$.
  This requires some elaboration. Eventually a poly machine (in the BGS sequence) will produce a satisfying line for x as its output given x as input. The upper bound for the machine with that ability is given by the first BGS index so that the code for x is smaller than the value at that index of the counterexample function.
  That means: we arrive at a machine M_m which outputs a correct satisfying line up to x as an input, and then begins to output wrong solutions.
- Alternatively check for $\mathsf{V}(x, 0)$, $\mathsf{V}(x, 1)$, …up to—if it ever happens—some s so that $\mathsf{V}(x, s) = 1$; or,
- Now, if f_c is fast-growing, then as the operation time of P_j is bounded by $|x|^k + k$, we have that $k \leq j$, and therefore it grows as $O(\mathsf{f}_c^{-1}(x))$. This will turn out to be a very slowly growing function.
  Again this requires some elaboration. The BGS machines are coded by a pair $\langle m, k\rangle$, where m is a Turing machine Gödel index, and k is as above. So we will have that the index j by which we code the BGS machine among all Turing machines is greater than k, provided we use a monotonic coding.
  More precisely, it will have to be tested up to j, that is the operation time will be bounded by $\mathsf{f}_c^{-1}(x)(|x|^{\mathsf{f}_c^{-1}(x)} + \mathsf{f}_c^{-1}(x))$.
  Again notice that the BGS index $j \geq k$, where k is the degree of the polynomial clock that bounds the poly machine.

10.5 Almost Maymin–Efficient Markets

We will now discuss the following:

Proposition 10.5.1 *If $P < NP$ isn't proved by primitive recursive arithmetic then there are almost Maymin efficient markets.* □

For a theory S with enough arithmetic—we leave it vague; we'll specify how much arithmetic in the example we'll soon discuss—and with a recursively enumerable set of theorems, for any provably total recursive function h there is a recursive, total, function g so that g dominates h.

Suppose now that we conjecture: the formal sentence $P < NP$ isn't proved by Primitive Recursive Arithmetic. Then the counterexample function f_c will be at least of the order of growth of Ackermann's function [3]. By the previous discussion about

[4]The BGS machine set is a set of time-polynomial Turing machines which includes algorithms that mimic all time-polynomial Turing machines. See above and check [1].

O'Donnell's algorithm, we see that the slow–growing exponential that bounds the operation time of the algorithm will be at least of the order of growth of the inverse function of Ackermann's function.

Given that condition, we can state:

Proposition 10.5.2 *If $P < NP$ isn't proved by Primitive Recursive Arithmetic then there are almost Maymin–efficient markets.* □

Comments

For details, [3]. We've briefly remarked that our proof, while restricted to a very particular situation, is in fact adequately general. That is the case: for a more general set of objects (policies etc.) has to imply ours, as our domain of objects would be a subset of the enlarged domain.

Also we require very little in our discussion—main tool is Post's theorem. As long as it holds, so does our proof. Does it hold for paraconsistent logics? That's an open question, which classes of paraconsistent logics would allow a proof of Maymin's beautiful theorem.

Acknowledgments This paper was supported in part by CNPq, Philosophy Section Grant no. 4339819902073398. It is part of the research efforts of the Advanced Studies Group, Production Engineering Program, at COPPE–UFRJ and of the Logic Group, HCTE–UFRJ. We thank Profs. R. Bartholo, S. Fuks (in memoriam), S. Jurkiewicz, R. Kubrusly, and F. Zamberlan for support.

References

1. Baker, T., Gill, J., Solovay, R.: Relativizations of the $P = ?NP$ question. SIAM J. Comput. **4**, 431–442 (1975)
2. Ben–David, S., Halevi, S.: On the independence of P *vs*. NP. Technical Report # 699, Technion (1991)
3. da Costa, N.C.A., Doria, F.A.: On the O'Donnell algorithm for NP–complete problems. Rev. Behav. Econ. (2016)
4. Maymin, P.Z.: Markets are efficient if and only If $P = NP$. Algorithmic Finance, **1**(1), 1 (2011)
5. O'Donnell, M.: A programming language theorem which is independent of Peano arithmetic. In: Proceedings of 11th Annual ACM Symposium on the Theory of Computation, pp. 176–188 (1979)
6. Post, E.L.: Introduction to a general theory of elementary propositions. Am. J. Math. **43**, 163 (1921)